# System Failure Prediction through Rare-Events Elastic-Net Logistic Regression

José M. Navarro, Hugo A. Parada G., Juan C. Dueñas

*Abstract*—**Predicting failures in a distributed system based on previous events through logistic regression is a standard approach in literature. This technique is not reliable, though, in two situations: in the prediction of rare events, which do not appear in enough proportion for the algorithm to capture, and in environments where there are too many variables, as logistic regression tends to overfit on this situations; while manually selecting a subset of variables to create the model is error-prone. On this paper, we solve an industrial research case that presented this situation with a combination of elastic net logistic regression, a method that allows us to automatically select useful variables, a process of cross-validation on top of it and the application of a rare events prediction technique to reduce computation time. This process provides two layers of cross-validation that automatically obtain the optimal model complexity and the optimal model parameters values, while ensuring even rare events will be correctly predicted with a low amount of training instances. We tested this method against real industrial data, obtaining a total of 60 out of 80 possible models with a 90% average model accuracy.**

***Online Failure Prediction; Machine Learning; System Management; Automatic Feature Selection; Logistic Regression; Multivariable Prediction***

## I. INTRODUCTION

The management of distributed systems and big networks is hard and time-consuming. When these systems are critical, e.g. they provide a core service for consumers of an organization, their well-being and performance turns into a key need. In these contexts, just assuring their reliability and minimizing the time they stay down is not enough. A proactive approach to failures is needed. This is where a technique called "Proactive Fault Management" comes into play. This discipline spans from knowing when a failure is going to happen to applying measures to avoid that failure and the preparation of repair mechanisms to minimize the harm this crash causes. Formally, these actions can be divided in four different steps[1]:

1. *Online Failure Prediction*: the identification of situations in the system that will possibly cause a failure. Usually, this is done through data mining and machine learning techniques.
2. *Diagnosis*: to pinpoint the location and/or cause of the predicted failure in step 1.
3. *Action Scheduling*: definition of which measures to take to respond to the forecasted failure.
4. *Execution of actions*: the actual execution of the scheduled plan in step 3.

This paper's contribution is framed in the first step of the four shown above, Online Failure Prediction. In it, we describe an offline-trained method we have developed in order to forecast online failures in a Spanish bank IT infrastructure network. This method uses Elastic Net Logistic Regression [2] as its base algorithm, overlays it with two layers of cross-validation for the automatic optimal tuning of the model's parameters and applies a rare events prediction technique inspired from [3] to drastically reduce the computation time of the models' training. The use of the elastic net allows us to automatically select the best features for each model, so we end up with a probably optimal model in terms of parameters and complexity, trained with a significantly low amount of instances.

On the rest of the paper we describe the current trends in Online Failure Prediction, the problem at hand, our experimentation and the available data. After that, we explain our proposed solution and algorithm in detail, the results obtained applying it to the available data and expose future lines of work that derive from this paper are drawn.

## II. ONLINE FAILURE PREDICTION

Salfner, Lenk and Malek describe in [1] a taxonomy englobing all the possible online failure prediction techniques based on the input data the method uses. But before doing so, it is appropriate to clarify the terminology they use:

- *Failure*: the manifestation of a deviation from the correct service, this is, a behaviour that can be observed.
- *Error*: the situation when the system deviates from the standard service.
- *Fault*: the supposed cause of an error.

Following these definitions, a fault causes an error, which can be observed through the consequent failures. Now, the taxonomy is:

- *Failure Tracking*: to predict future failures based on the appearance of previous failures.
- *Symptom Monitoring*: to base the prediction of failures in the periodic monitoring of system

variables, such as the resource consumption or network usage.

- *Detected Error Reporting*: the prediction of future failures based on the analysis of previous errors happened in the system stored in error logs. The difference between this and failure tracking is that this prediction is based on previous errors, not failures.
- *Undetected Error Auditing*: to proactively search the system for an incorrect undetected state, this is, an error and, based on them, to forecast the occurrence of future failures.

After explaining the different possibilities, let us study some relevant works on online failure prediction. It must be noted that this discipline also involves the required data pre-processing to ease or facilitate the creation of predictive models. For example, in [4], Yu et al. propose a filtering method that performs feature selection and instance selection to eliminate noisy and redundant data and in [5] Zheng et al. present a three step pre-processing method that includes the categorization of events, the removal of temporal and spatial redundant records and the use of the a priori algorithm to find correlated events that are actually related to just one cause. In the area of actual failure prediction, there are several recent papers on the issue. In 2006, Liang et al. [6] propose a series of three ad-hoc created predictors for the Blue Gene supercomputer, based on the analysis of the failure characteristics found on its logs. Watanabe et al. propose a method quite similar to association rules in [7]: the creation of an event dictionary with the associated probabilities for each entry to precede a failure. On the other hand, in [8], Sonoda, Watanabe and Matsumoto show how to identify patterns that lead to system failures through Bayesian learning, achieving a precision of over 0.8 and recall over 0.7. The creation of rules is again proposed in [9], this time through fault tree analysis, which receives as inputs the outputs of an ARMA (Autoregressive Moving Average) time series model [10] that uses the resources consumption metrics of hosts and transforms it into a series of boolean conditions which, when activated, imply the appearance of a failure cause in the system. This is the first instance of a combination of methods, a popular choice for researchers trying to model a complex system. Generally, when trying to do so, there are three main options:

1. To use an ensemble method, this is, to combine the output of several methods to form the actual prediction. This is the basis of random forests [11].
2. To use a complex algorithm, such as an Artificial Neural Network [12] or a Support Vector Machine, that is able on its own to completely model the system.
3. To concatenate simple algorithms that cover several phases of the process, such as the one we saw in [9], where the available, unlabeled data, are first transformed into labeled data by an ARMA model and then the actual prediction is created through a fault tree.

Several more complex or not commonly used algorithms have also been successfully applied to Online Failure Prediction, such as a genetic algorithm [13] applied to Blue Gene logs or semi-hidden Markov chains [14], a technique usually applied to natural language processing or genetic sequencing analysis, where the sequence of errors in the system and the time between them are analysed to forecast the appearance of a future error. A similar, parallel approach to the one shown in [9] is found in [15], where Guan, Zhang and Fu propose the use of Bayesian methods to detect anomalies in the system, followed by the validation of an expert and the creation of a decision tree based on the created labelled data.

There are also proposals for two different phases of the prediction process. The first one, found on [16], dealing with the data collection phase, shows how to find the features that are more highly correlated with the appearance of system failures by the injection of realistic software failures in the system, which in turn allows for the reduction of the dataset to collect. On the other hand, [17] exposes two possibilities for the practical implementation of an online failure predictor, period-based and event-based, and proves how an event-based approach shows better results than a period-based one.

In this context, we propose a method that falls into the Detected Error Reporting category and employs a simple algorithm (logistic regression), combined with two three different overlays (cross validation, regularization and rare events prediction techniques). In contrast with [16], our algorithm is able to automatically detect which features are correlated with the output without direct manipulation of the system. Furthermore, it will be shown in section V how the results it obtains are on par or even better that the discussed works.

## III. SCENARIO

The problem we were proposed to solve is as follows: we were given data from the network hosts of a part of a bank application infrastructure. This hosts were part of an intranet and an internet, each of them containing:

- Eighteen switches.
- Two DNS.
- Four routers.
- Six firewalls.
- Six load balancers.

Making a total amount of thirty six nodes. For each one of these nodes, we had the following data source:

- *System events*: log files for the system, containing all the events that happened on it, classified by event severity, node they happened in, date and event type. Specifically, the severity field contained four different values: “critical”, “major”, “minor”, and “blank”.

With these data, the objective of our project was to be able to forecast correctly the occurrence of fatal failures within a certain time frame Events with a severity value of "critical" were considered fatal failures. Additionally, if we were able to correctly model the critical events, being able to correctly model events with "major" severity would be a plus. The measures we would use to evaluate the obtained models would be precision and recall, alongside the f-score[18]. A minimum f-score of 0.85 was the condition to consider a created model a success.

## IV. Approaches and Algorithm Proposal

In this section we will narrate the different approaches we followed to fulfil the project's objective, the problems we encountered and what we did to overcome them. Our philosophy was to start from simple models and increase their complexity until the desired results were obtained.

As a first test, we tried to predict critical events on any point of the system (from now on, just failures) based on the previous appearance of any event on any point of the system. Thus, we did not plan to include node-awareness in our model yet. We used four months of data, the first three for training and the fourth one as a test set. Again, following the terminology established in [1], we used as input data the presence or absence of events in an observation window of time "t", where "t" equals 5 or 30 minutes in our experiment and a prediction window of time "t′", equalling 5 or 30 minutes too. We did not apply any offset to the prediction window, a situation that could change in future experiments. Combinations of all possibilities yield four different window sets for each modelled event. We decided to deal with this task by constructing models of increasing complexity, as simpler models are desired over complex ones on equal performance if there is enough data to feed the model with, which we had. Following this basis, we first created the single variable models, as proposed by Zumel and Mount in [19]. This kind of models tries to check if there is any direct relationship between categorical variables by creating a confusion matrix between the independent variable we want to test and the target, dependent variable. Apart from their simplicity, it is recommended to always build them first when trying to tackle a data science problem because they establish a minimum performance value more complex predictors have to surpass to be considered useful. The obtained results were not satisfactory, with most variables showing no relationship and with lower than 0.5 f-scores for the ones that did show some relationship at all.

The next step up in complexity was to create logistic regression models for each independent variable on its own. Logistic Regression is a simple method, widely used in binary classification problems such as the one we were dealing with. It takes an input "x", ponders it by a set of weights "w" and applies a sigmoid function (defined in (1)) to the result

$$\text{sigmoid}(x)=\frac{1}{1+\exp(-x)} \tag{1}$$

obtaining a number between 0 and 1 which can be used as a direct probability of the output or converted in a binary output by defining a numeric threshold. We used the sigmoid function as it is the form derived from the Bernouilli distribution exponential family. A logistic regression model includes a training phase in which the model parameters, the weights "w" are optimized through a specific method, being gradient descent the most popular one, which seeks to minimize the cost function, normally defined as

$$C(\mathbf{x})=\frac{1}{2}\sum_{i=1}^{N}(y-\boldsymbol{w}^T\mathbf{x})^2 \tag{2}$$

Again, we tried testing every independent feature against each possible outcome, searching for direct relationships. The results obtained this time were definitely better than the obtained with single variable models, showing that the training phase had had effect, but they were still not enough, as the maximum f-score we obtained was just 0.75.

At this phase we can summarize the main problem we found while creating both the single variable models and the logistic regression ones: the process of finding significant input variables is slow, cumbersome and manual for both methods, as it implies testing every possible variable against each possible target. One possible way to overcome this would be to just fit each logistic regression model with just all the possible input features at once. This is not recommended, though, because this would most surely lead to an overfitted model. So we decided to increase the complexity of the solution one step and use methods that would allow us to, at least, fit models with a large number of features and, ideally, automatically select the significant features for a logistic regression model. One of the existing possibilities is to add a regularization term to the cost function of logistic regression, (2). There are two common types of regularization. The first one is lasso [20], which forces the L1-norm of the weight vector to be less than a specified constant, β:

$$|\boldsymbol{w}|_1=\sum_i |w_i| < \beta \tag{3}$$

The second one is called ridge regression [21], and it is similar to the lasso, but it changes the L1-norm penalization term for the L2-norm

$$|\boldsymbol{w}|_2=\sum_i |w_i|^2 < \beta \tag{4}$$

This presented another problem: which regularization technique should we use? As we had almost 80 different models to create, studying manually each one was not a feasible option. So we had to be able to apply the optimal regularization term to each model automatically. This is why we chose to use a third regularization technique called Elastic Net, defined in [2]. This method adds a

hyperparameter $\alpha$ that controls the regularization term and fixes the amount of lasso penalization and the amount of ridge regression penalization to be applied to each problem. The full cost function for elastic net logistic regression is then

$$C(\mathbf{x})=\frac{1}{2}\sum_{i=1}^{N}(y-\boldsymbol{w}^T\boldsymbol{x}_i)^2+\alpha|\boldsymbol{w}|_1+(1-\alpha)|\boldsymbol{w}|_2 \qquad (5)$$

Out of the two possible basic regularization methods, lasso is able to bring features' weights to zero, which allows it to effectively perform feature selection, so we decided to fix $\alpha$ to 1 (full lasso) to perform our first tests with the Elastic Net. These first tests returned a sort of mixed result. Out of 80 possible models, only 12 had an f-score higher than 0. But, the ones that had, obtained a really high score, almost always higher than 0.9. This suggested that the elastic net approach was a suitable one, but there was something wrong with the input data. So, after inspecting it, we drew two important conclusions:

- The event distribution was not constant at all, so several events only happened on certain months. A huge amount of events was not present on the test period, so it was impossible to test the created model's performance.
- Out of almost 18700 training instances, there were very few instances when the target event actually appeared.

Added to this, we still had to decide which regularization mixture use, as we had just fixed alpha to a constant 1. So we set out to solve these three problems. To deal with the scarcity of training instances, we relied on the advices and techniques laid out in [3], where King and Lanche state how a large amount of zeros (instances with the absence of target event) does not add useful information to the model and thus, can be pruned out of the training instances. We combined this with a transformation of the input data and ended up with a data pre-processing algorithm that created the actual input for the model. It is similar, conceptually, to what 5-fold cross validation does to data, but with three different datasets and without the repetition of cross validation (data are not reordered and tested again several times). As a first iteration, we fixed the proportion of data split into each dataset as a design choice. The algorithm steps are:

- Given a dataset of length K.
- Separate the N instances where the target event appears (1's) into set A from the M ones it does not (0's) into set B, having N+M=K.
- Randomize the order of both sets.
- Take the first 0.6N of A and mix it with the first 0.6$\beta$N of B, being $\beta$ a user-defined constant. This creates the Training Dataset.
- Take the next 0.2N of A and mix it with the next 0.2$\beta$N of B. This creates the Validation Dataset.
- Take the last 0.2N of A and mix it with the next 0.2$\beta$N of B. This creates the Test Dataset.

If, at any moment, there are not enough instances in B to fill any dataset, B is sampled with repetition.

This algorithm for the pre-processing of the input data solves the first two problems: it ensures that the target event will be present in each dataset and it also adapts the number of training instances to the number of times the target event appears, optimizing the computation time.

To cover the last problem, 8 models are created for each target event. These models are identical except for their regularization mixture parameter value, $\alpha$, and they cover the whole range of mixtures, including full lasso and full ridge ($\alpha$ equal to 1 and equal to 0) and values close to them, as the authors state how these eliminate the weaknesses of each regularization method. We now describe the algorithm we used to create a model for each of the E target events data:

- Separate the dataset in three splits using algorithm 1.
- Using the training dataset, train eight different models using k-fold cross validation each model's complexity. Each model uses a different value of alpha, namely, {1, 0.98, 0.8, 0.6, 0.4, 0.2, 0.05, 0}.
- Take each model and test it against the validation dataset.
- Compute each model's f-score.
- Take the best scored model and test it against the test dataset.
- Use the final f-score as the model's metric for validation purposes.

This model creation process is shown graphically in Fig. 1, where the top row shows the conceptual process and the bottom row states the actual process. What we do is, effectively, use two layers of cross-validation to automatically tune each model's complexity and use the optimal regularization mixture. All of this, using an optimal dataset in terms of size and instance distribution (taking into account that the percentages by which we divide the datasets and the proportion of zeros we use is, at the time, an *ad-hoc* assumption). This way, we solve the three problems stated before and, furthermore, it allows to detect which features have the most influence in each model training each model once and without the danger of overfitting. In the next section we describe thoroughly the results we obtained using this model.

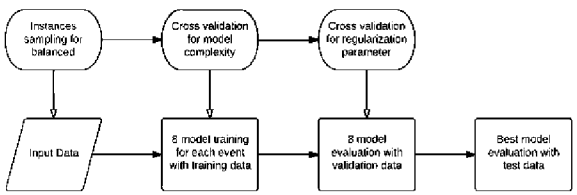


Figure 1. Proposed model creation algorithm

## V. RESULTS

As we previously stated, we trained models for four different observation and prediction windows combinations. A detailed summary of these models performance can be found on Table 1 (where observation windows are named as "B" for "backwards" and prediction windows are named "F" for "forward"). On it we see how the number of successfully created models has increased drastically from the elastic net preliminary test we described in the previous chapter (12 non-zero models). Apart from that, the precision and recall average values is very high, which yields an f-score higher than the threshold we set before.

To study the effect of the four different window combinations, on Fig. 2 we can see every model's precision plotted against its recall, which gives us an insight about the spread of the created models. Studying it we see that the observation window has little effect on the spread of the models (though, looking at table 1 we can check that it captured less models than the 5-minute observation one). On the other hand, the prediction window has a clearer, drastic effect on the spread of the models, driving the f-score down in almost 0.8 points. These two insights suggest that the system does not have memory, this is, that only the most recent events influence the posterior events, and that increasing the prediction window adds random noise that pollutes the training phase.

Recalling our first statements, the initial objective was to be able to forecast critical-severity events. Specifically, there were three different type of events in this system. Taking the best result of the created models, their score can be found on table 2. It is, clearly, a very good result. Out of 6 created models, 5 offer an outstanding result.

The obtained results were so good that we started to suspect some overfitting problem. To check it was not so, we followed the guidelines established in [22] and studied whether our sample sizes for each model was higher than 10 instances per non-zero variable. The results showed that more than 90% of them are higher than the established threshold, which marks the minimum amount of needed instances. The ones that are not over that line should be treated cautiously, as they may be overfitted. Furthermore,

TABLE I. MODELS PERFORMANCE

| Metric | Window combinations | | | |
|---|---|---|---|---|
| | ***B5/F5*** | ***B30/F5*** | ***B5/F30*** | ***B30/F30*** |
| Models | 78 | 78 | 77 | 77 |
| Non-zero models | 56 | 52 | 60 | 59 |
| Average precision | 0.949 | 0.933 | 0.882 | 0.882 |
| Average recall | 0.888 | 0.897 | 0.811 | 0.823 |
| Average f-score | 0.908 | 0.906 | 0.823 | 0.830 |
| F-score standard deviation | 0.124 | 0.116 | 0.203 | 0.223 |

TABLE II. CRITICAL EVENTS MODELS PERFORMANCE

| Critical Events | Critical Event Predictor F-score | |
|---|---|---|
| | ***5-minute prediction*** | ***30-minute prediction*** |
| Event 1 | 0.94 | 0.92 |
| Event 2 | 1 | 0.66 |
| Event 3 | 0.99 | 1 |

we have two powerful reasons to think our models are able to generalize correctly:

- The use of regularization is a powerful tool against overfiting.
- The splitting of the dataset in three parts and the final testing on unseen data is a guarantee that the model works correctly with new data.

We should raise one word of caution, though. These models work correctly when the network functions on its own, but they would not be able to forecast changes not caused by the network itself, such as configuration changes or failures produced by the applications running over the network, as they wouldn't leave any trace in the previous event records for the model to detect.

## VI. CONCLUSIONS AND FUTURE WORK

To sum up, we think that the model creation algorithm we propose, making usage of a combination of elastic net regularization, rare events prediction techniques and two layers of cross-validation is a powerful tool for forecasting failures in a distributed system. We have shown how our algorithm allows for the automatic tuning of each model's complexity and regularization mixture, while ensuring that how the data are distributed will not affect the model creation.

There are several lines of work that could span from this paper. A direct one is to improve and tune all the fixed parameters and assumptions taken over the proposed method in this paper. We would like, also, to test our data pre-processing algorithm to test its stability. It would be very interesting too to compare in complexity, performance and training time the proposed algorithm with standard state of the art complex algorithms, such as Random Forests, Artificial Neural Networks, Support Vector Machines or Ensemble Methods.


## ACKNOWLEDGMENT

We thank Produban, Ana Borrego, Francisco Guirado and his team for providing us with data to analyse and their support and collaboration in this research.


## REFERENCES


[1] Salfner, F., M. Lenk, and M. Malek. "A Survey of Online Failure Prediction Methods." *ACM Computing Surveys (CSUR)* 42, no. 3 (2010): 10

[2] Zou, Hui, and Trevor Hastie. "Regularization and Variable Selection via the Elastic Net." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67, no. 2 (2005): 301–20.

[3] King, Gary, and Langche Zeng. "Logistic Regression in Rare Events Data." *Political Analysis* 9, no. 2 (2001): 137–63.

[4] Yu, Li, Ziming Zheng, Zhiling Lan, T. Jones, J.M. Brandt, and A.C. Gentile. "Filtering Log Data: Finding the Needles in the Haystack." In *2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 1 –12, 2012.

[5] Zheng, Ziming, Zhiling Lan, B.H. Park, and A. Geist. "System Log Pre-Processing to Improve Failure Prediction." In *IEEE/IFIP International Conference on Dependable Systems Networks, 2009. DSN '09*, 572 –577, 2009.

[6] Liang, Y., Y. Zhang, M. Jette, A. Sivasubramaniam, and R. Sahoo. "BlueGene/L Failure Analysis and Prediction Models." In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, 425–34, 2006.

[7] Watanabe, Y., H. Otsuka, M. Sonoda, S. Kikuchi, and Y. Matsumoto. "Online Failure Prediction in Cloud Datacenters by Real-Time Message Pattern Learning." In *2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, 504–11, 2012.

[8] Sonoda, M., Y. Watanabe, and Y. Matsumoto. "Prediction of Failure Occurrence Time Based on System Log Message Pattern Learning." In *2012 IEEE Network Operations and Management Symposium (NOMS)*, 578–81, 2012.

[9] Chalermarrewong, Thanyalak, Tiranee Achalakul, and Simon Chong Wee See. "Failure Prediction of Data Centers Using Time Series and Fault Tree Analysis." In *2012 IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS)*, 794 –799, 2012.

[10] Whittle, P. (1951). *Hypothesis Testing in Time Series Analysis*. Almquist and Wicksell. Whittle, P. (1963). *Prediction and Regulation*. English Universities Press. ISBN 0-8166-1147-5.

[11] Guan, Qiang, Ziming Zhang, and Song Fu. "A Failure Detection and Prediction Mechanism for Enhancing Dependability of Data Centers." *International Journal of Computer Theory and Engineering* 4, no. 5 (2012).

[12] Smoczek, Jaroslaw. "The Survey of Soft Computing Techniques for Reliability Prediction." *Journal of KONES* 19 (2012): 407–14.

[13] Zheng, Z., Z. Lan, R. Gupta, S. Coghlan, and P. Beckman. "A Practical Failure Prediction with Location and Lead Time for Blue Gene/p." In *Dependable Systems and Networks Workshops (DSN-W), 2010 International Conference on*, 15–22, 2010.

[14] Salfner, Felix, and Miroslaw Malek. "Using Hidden Semi-Markov Models for Effective Online Failure Prediction." In *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*, 161–74. IEEE, 2007.

[15] Guan, Q., Z. Zhang, and S. Fu. "Proactive Failure Management by Integrated Unsupervised and Semi-Supervised Learning for Dependable Cloud Systems." In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, 83–90, 2011.

[16] Irrera, I., J. Durães, M. Vieira, and H. Madeira. "Towards Identifying the Best Variables for Failure Prediction Using Injection of Realistic Software Faults." In *2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing (PRDC)*, 3 –10, 2010.

[17] Yu, Li, Ziming Zheng, Zhiling Lan, and S. Coghlan. "Practical Online Failure Prediction for Blue Gene/P: Period-Based vs Event-Driven." In *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 259 –264, 2011.

[18] Van Rijsbergen, C. J. (1979). *Information Retrieval* (2nd ed.). Butterworth.

[19] Nina Zumel and John Mount. 2014. *Practical Data Science with R* (1st ed.). 18 - 123. Manning Publications Co., Greenwich, CT, USA.

[20] Tibshirani, R. (1996). "Regression shrinkage and selection via the lasso". Journal of the Royal Statistical Society, *Series B* **58** (1): 267–288.

[21] Tikhonov, Andrey Nikolayevich (1943). "Об устойчивости обратных задач" [On the stability of inverse problems]. *Doklady Akademii Nauk SSSR* **39** (5): 195–198.

[22] Peduzzi, P., Concato, J., Kemper, E., Holford, T. R., & Feinstein, A. R. (1996). A simulation study of the number of events per variable in logistic regression analysis. *Journal of clinical epidemiology*, *49*(12), 1373-1379.


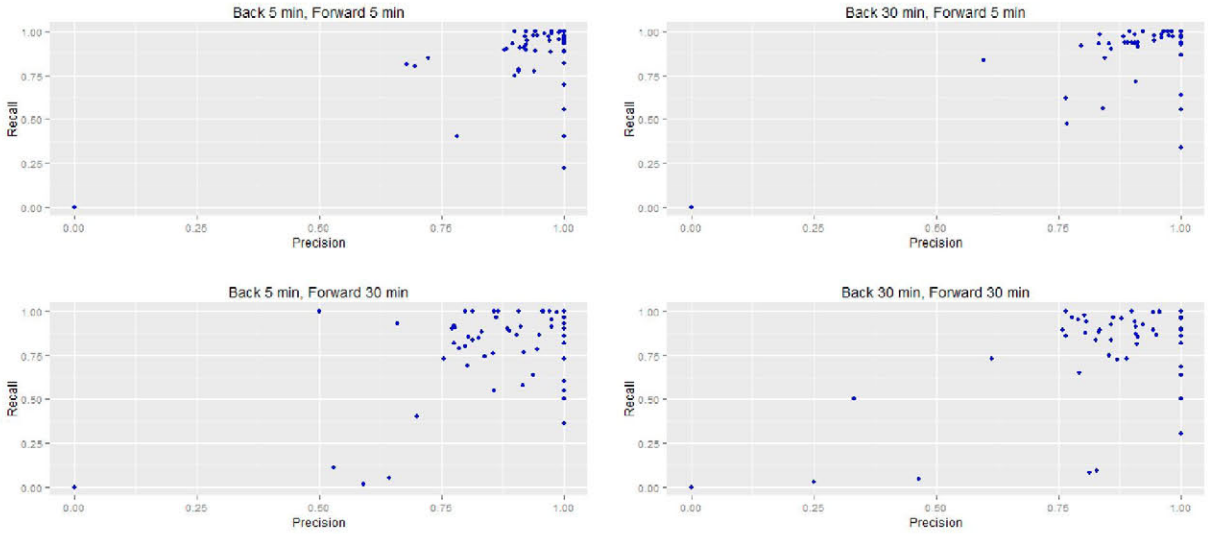


Figure 2. Created Models Performance